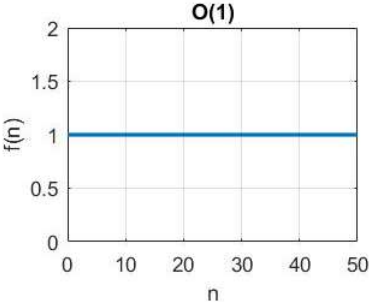
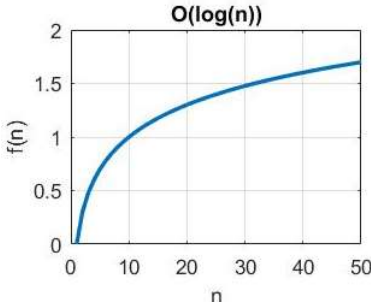
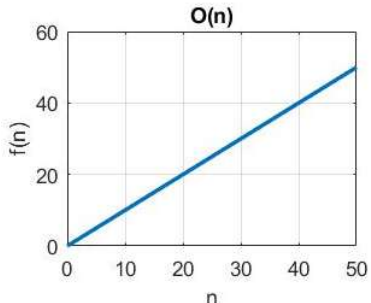
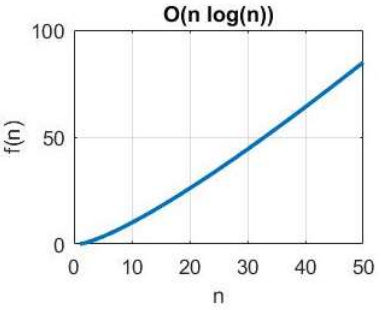
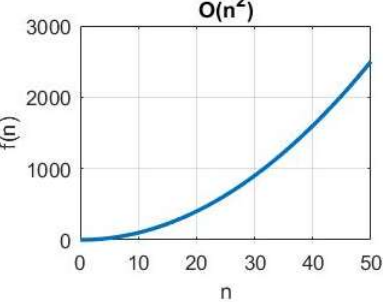


**Złożoność obliczeniowa**- oszacowanie czasu wykonania algorytmu. Mierzmy liczbę operacji, następnie szukamy funkcji opisującej liczbę operacji w zależności od danych wejściowych. Notacja O jest szacowaniem z góry. Ponieważ chcemy tylko przybliżyć wartość, pomijamy wszelkiego rodzaju stałe, które nie mają znaczenie przy dużych n. Zatem  $O(2n + 5)$ ,  $O(2n)$  i  $O(n)$  są uznawane za złożoność obliczeniową  $O(n)$ .

		PRZYBLIŻONE WYKRESY	PRZYKŁADOWY KOD	OPIS
1	$O(1)$		<p>źródło: <a href="http://www.samouczekprogramisty.pl/podstawy-zlozonosci-obliczeniowej/">http://www.samouczekprogramisty.pl/podstawy-zlozonosci-obliczeniowej/</a></p> <pre> public int sum(int[] numbers) {     if (numbers == null    numbers.length == 0) {         return 0;     }     return (numbers[0] + numbers[numbers.length - 1]) * numbers.length / 2; } </pre>	<p>Jest to tzw. złożoność stała, która jest niezależna od liczby danych wejściowych. Przy obliczeniu sumy ciągu arytmetycznego (kod po lewej), nie iterujemy po wszystkich elementach tablicy, zatem czas wykonania jest stały i niezależny od wielkości tablicy.</p>
2	$O(\log n)$		<pre> public boolean binarySearch(int[] numbers, int number) {     int indexLow = 0;     int indexHigh = numbers.length - 1;     while (indexLow &lt;= indexHigh) {         int indexMiddle = indexLow + (indexHigh - indexLow) / 2;          if (number &lt; numbers[indexMiddle]) {             indexHigh = indexMiddle - 1;         }         else if (number &gt; numbers[indexMiddle]) {             indexLow = indexMiddle + 1;         }         else {             return true;         }     }     return false; } </pre>	<p>Jest to tzw. złożoność logarytmiczna, której czas wykonania zależy od wyniku logarytmu z wielkości danych wejściowych. Przykładowy kod to przeszukiwanie binarne posortowanej tablicy, tego typu przeszukiwanie charakteryzuje się tym, że w każdym kroku tablica jest dzielona na dwie części (zmniejszany jest zbiór do przeszukiwania)</p>
3	$O(n)$		<pre> public int sum(int[] numbers) {     int sum = 0;     for (int number : numbers) {         sum += number;     }     return sum; } </pre>	<p>Jest to tzw. złożoność liniowa, której czas wykonania jest wprost proporcjonalny do ilości danych wejściowych. Przykładowo kod obliczający sumę wszystkich liczb w tablicy, wymaga użycia pętli i sprawdzenia wszystkich elementów tablicy, zatem musimy wykonać N kroków.</p>

4	$O(n \log n)$		<pre> public static int[] sort(int[] numbers) {     if (numbers.length &lt;= 1) {         return numbers;     }     int[] first = new int[numbers.length / 2];     int[] second = new int[numbers.length - first.length];     for (int i = 0; i &lt; first.length; i++) {         first[i] = numbers[i];     }     for (int i = 0; i &lt; second.length; i++) {         second[i] = numbers[first.length + i];     }     return merge(sort(first), sort(second)); }  private static int[] merge(int[] first, int[] second) {     int[] merged = new int[first.length + second.length];     for (int indexFirst = 0, indexSecond = 0, indexMerged = 0;         indexMerged &lt; merged.length; indexMerged++) {         if (indexFirst &gt;= first.length) {             merged[indexMerged] = second[indexSecond++];         }         else if (indexSecond &gt;= second.length) {             merged[indexMerged] = first[indexFirst++];         }         else if (first[indexFirst] &lt;= second[indexSecond]) {             merged[indexMerged] = first[indexFirst++];         }         else {             merged[indexMerged] = second[indexSecond++];         }     }     return merged; } </pre>	<p>Jest to tzw. złożoność liniowo-logarytmiczna, której czas wykonania jest wprost proporcjonalny do iloczynu danych wejściowych i ich logarytmu.</p> <p>Przykładowy kod to sortowanie tablicy przez scalanie. Dzielimy tablicę tak jak przy przeszukiwaniu binarnym na dwie części (N-razy) do czasu, aż każda z nich będzie miała długość 1. Następnie scalamy je ze sobą. Każde scalenie to koszt <math>O(n)</math>, a każde dzielenie tablicy na pół <math>O(\log(n))</math>.</p>
5	$O(n^2)$		<pre> public int[] sort(int[] numbers) {     for (int i = 0; i &lt; numbers.length; i++) {         for (int j = 0; j &lt; numbers.length - 1; j++) {             if (numbers[j] &gt; numbers[j + 1]) {                 int temp = numbers[j + 1];                 numbers[j + 1] = numbers[j];                 numbers[j] = temp;             }         }     }     return numbers; } </pre>	<p>Jest to tzw. złożoność kwadratowa, której czas wykonania jest wprost proporcjonalny do kwadratu ilości danych wejściowych.</p> <p>Przykładem jest sortowanie bąbelkowe, w zagnieżdżonych pętlach iterujemy N razy po N elementach tablicy.</p>

6	$O(2^n)$		<pre> public static int[][] powerSet(int[] numbers) {     int two_pow_n = 1 &lt;&lt; numbers.length;      int[][] powerSet = new int[two_pow_n][];     for (int subsetIndex = 0; subsetIndex &lt; two_pow_n; subsetIndex++) {         powerSet[subsetIndex] = pickNumbers(subsetIndex, numbers);     }     return powerSet; }  private static int[] pickNumbers(int subsetIndex, int[] numbers) {     int howManyOnes = 0;     int temp = subsetIndex;     while (temp &gt; 0) {         if (temp % 2 == 1) {             howManyOnes++;         }         temp &gt;&gt;= 1;     }      int[] subset = new int[howManyOnes];      for (int charIndex = 0, lastElementIndex = 0; subsetIndex &gt; 0; charIndex++)     {         if (subsetIndex % 2 == 1) {             subset[lastElementIndex++] = numbers[charIndex];         }         subsetIndex &gt;&gt;= 1;     }      return subset; } </pre>	<p>Jest to tzw. złożoność wykładnicza, czas wykonania rośnie wykładniczo względem ilości danych wejściowych.</p> <p>Kod przykładowy ma złożoność większą niż wykładnicza tj. <math>O(\log(n) 2^n)</math>. Na wejściu mamy tablicę unikalnych liczb, zwracamy tablicę zawierającą wszystkie podzbiory elementów tablicy wejściowej. Pętla w metodzie powerSet wywołana jest <math>2^n</math> razy, w pickNumber mamy dwie pętle o złożoności <math>\log(n)</math>.</p>
7	$O(n!)$		<p>Kod jest bardzo obszerny, dla chętnych do zobaczenia na stronie:  <a href="https://eduinf.waw.pl/inf/alg/001_search/0140.php">https://eduinf.waw.pl/inf/alg/001_search/0140.php</a></p> <p>Zanim uruchomicie wiedzcie, że złożoność jest ogromna i czas wykonania także jest duży, przykładowo na wykresach w kolumnie po lewej możecie zauważyć, że dla 50 danych wejściowych przy tej złożoności obliczeniowej uzyskujemy wartość <math>\sim 3 \cdot 10^{64}</math>, dla złożoności <math>O(n^2)</math> jest to 2500, a dla <math>O(n)</math> tylko 50.</p>	<p>Jest to złożoność typu silnia, czas wykonania rośnie z szybkością silni względem ilości danych wejściowych.</p> <p>Przykładem problemu jest problem komiwojażera, należy znaleźć najkrótszą trasę rozpoczynając od miasta A przechodzącą jednokrotnie przez wszystkie pozostałe miasta i wracającą do miasta A.</p>